Route Planning on the Map* and using PRAISE-HK Web Service to Provide Extra Information about Air Pollution Exposure Risk

* *Map inside Hong Kong only*

[***Note: this is not a basic level App Inventor tutorial. Readers are advised to write one or two basic level apps before trying this one.***]

PRAISE-HK[1] API is a web service for providing air quality information with high data-density[2] (down to street level) and high accuracy[3], and is able to provide 48-hour forecasted air quality information.

To access PRAISE-HK data, our team has developed an "extension[4]" - namely "PRAISE_HK_web" for users to access our data. This tutorial aims to demonstrate how to calculate the air pollution exposure risk based on the planned route. The exposure is obtained through an advanced PRAISE-HK web service which leverages the more basic level air quality information.

Approach Overview

1. We use a map to plan the route. The route is found by: first clicking two points on the map; then using a navigation component to plan a route between these two points.
2. After the route is obtained, we send it to the PRAISE-HK API to calculate the corresponding exposure.
3. After the results are returned (actual implementation is a series of results), we check if they are valid. Then displaying the value after a bit of manipulation.

Okay, if you are ready, let's get started!

Step 1. Getting an API Key

The route planning is provided by another web service called "openroute service". In order to use it, we have to go to its official site: https://openrouteservice.org/, to open an account and acquire an API key. So, first of all, we click the above link and enter the site.

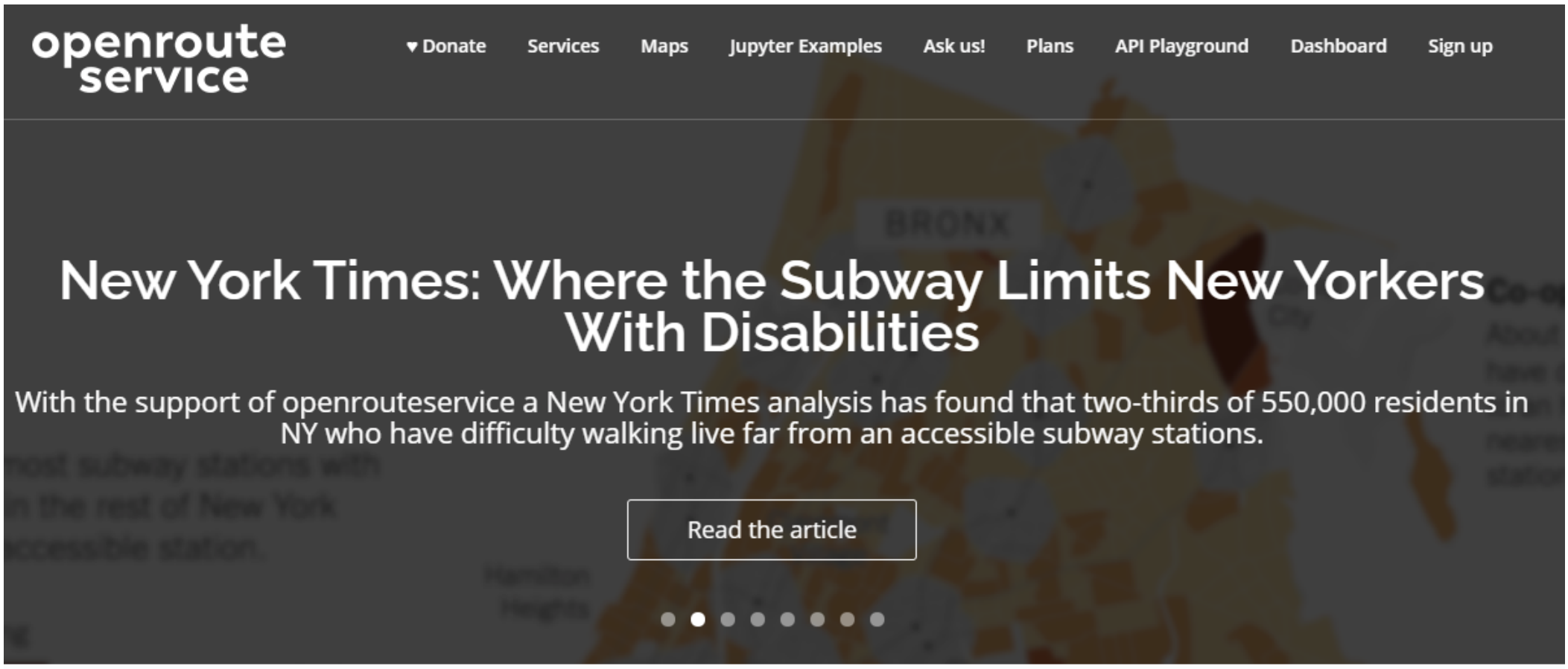
---

[1] **PRAISE-HK** is a short form for "Personalized Real-time Air-quality Informatics System for Exposure in HK" with the project goal to empower the public with personalized air quality information.

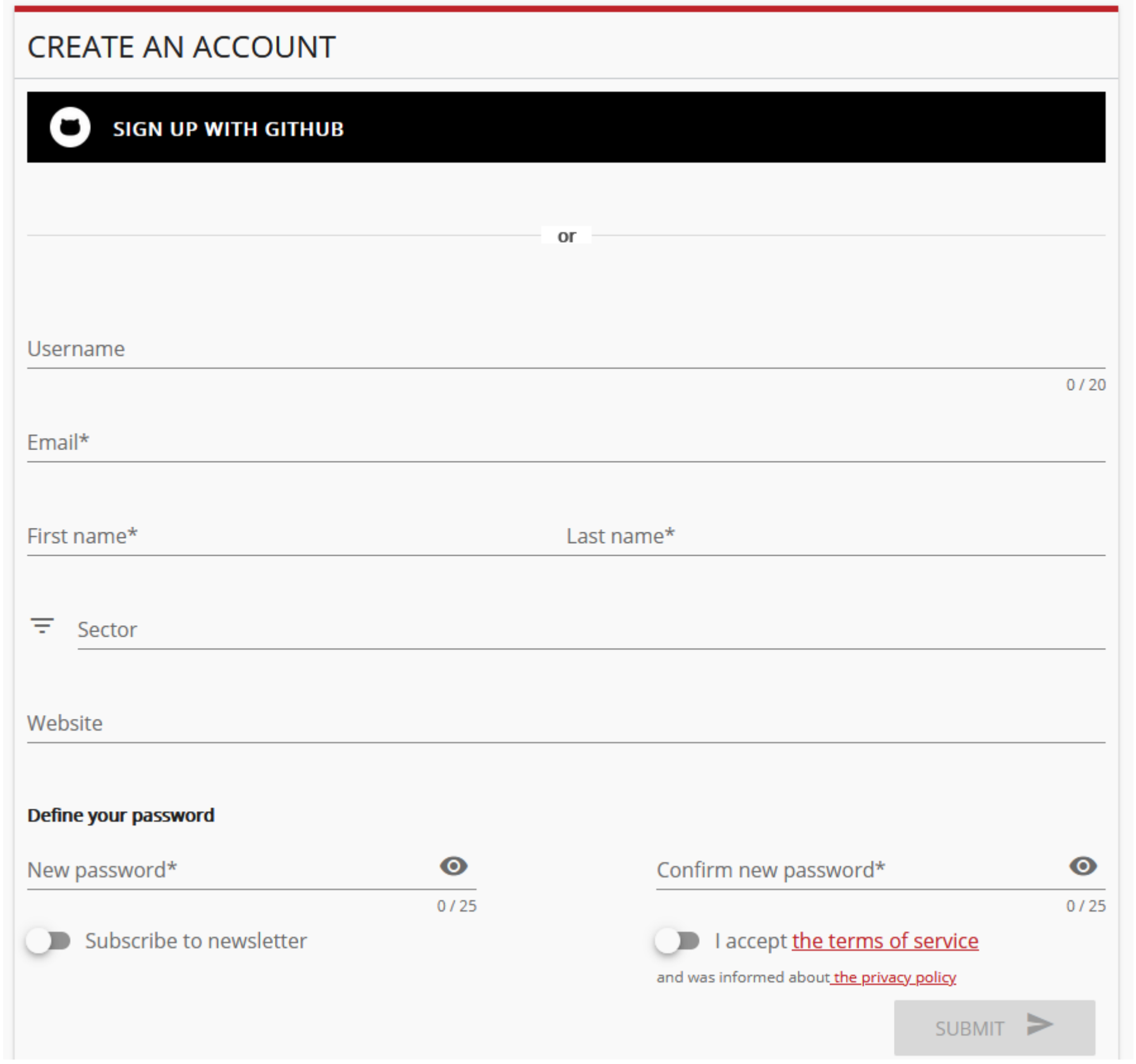
[2] **High data-density: PRAISE-HK** is able to provide air quality (and associated health risk) information up to 2-meter resolution.

[3] **How accurate are PRAISE-HK predictions compared with data from Hong Kong's air quality monitoring stations?**

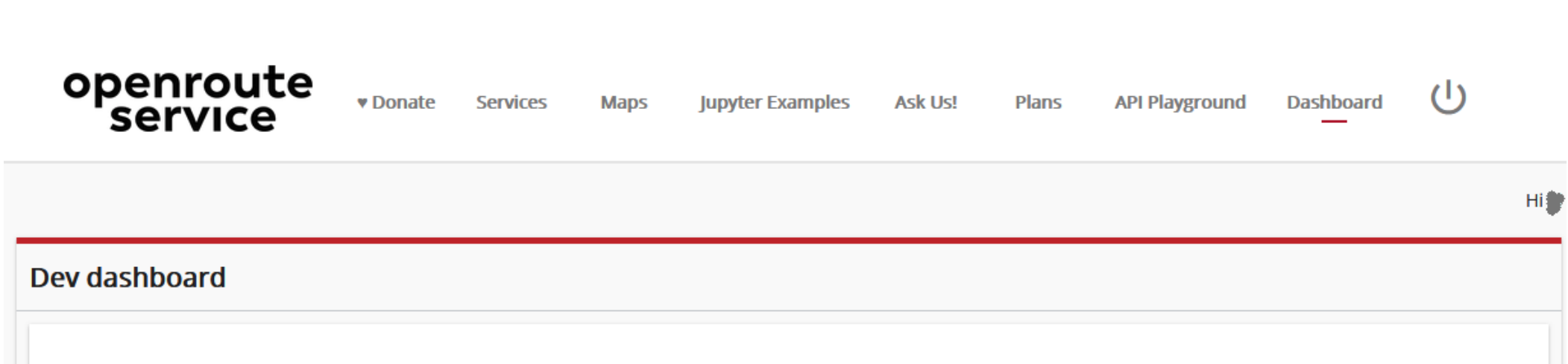
[4] **What is an "extension" here?** An "extension" provides app developers additional information/components for advanced and extended app features. Please refer to the following article for details.



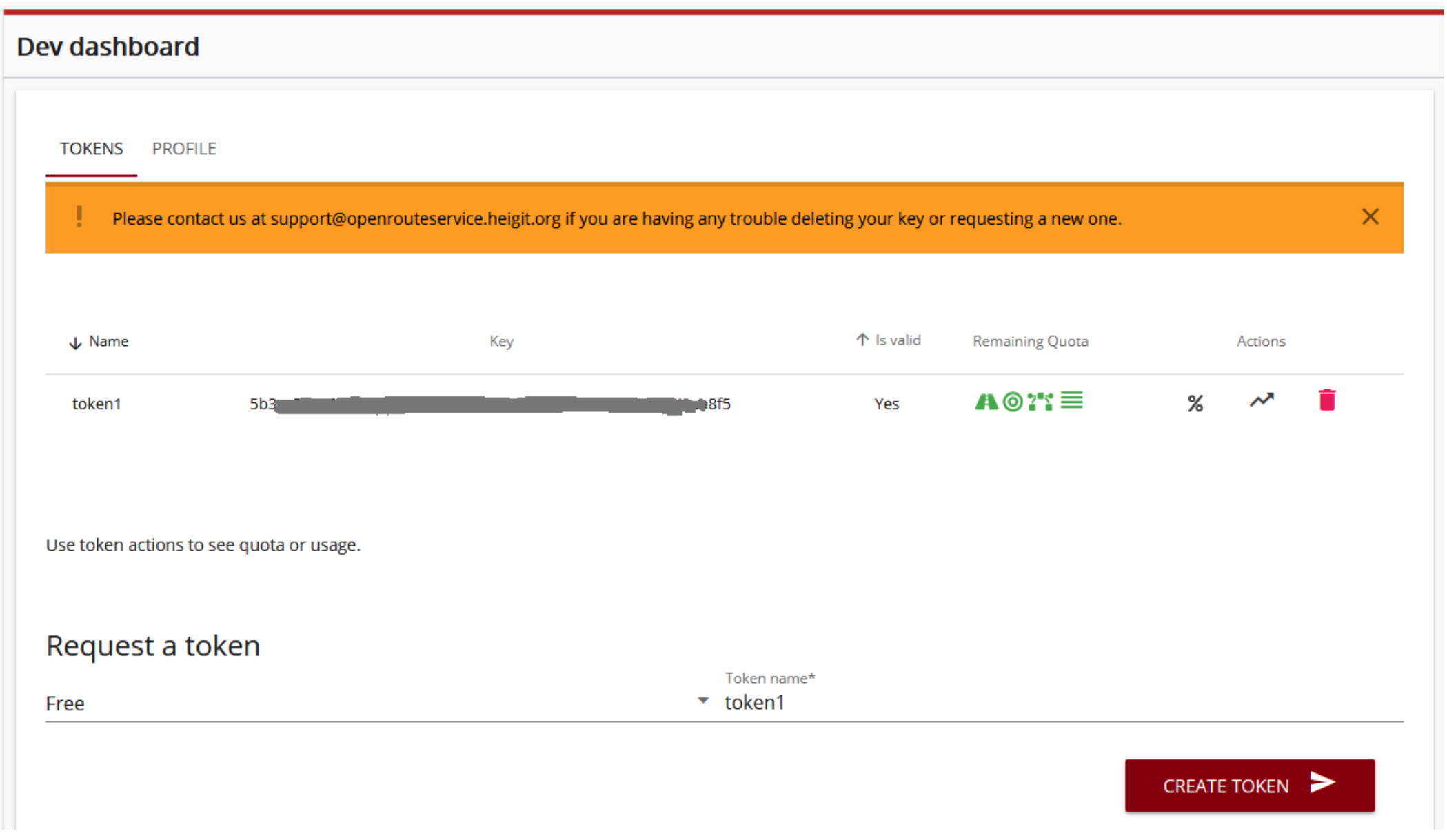
We then click the “Sign up” (which is at the top-right corner of the page) to create an account.



Fill in the required information and then use your registered email account (an activation email will be sent to this email account) to activate it. After activation and signing in, you will be brought to a “Dashboard” page’s “TOKENS” tab.



At the bottom of the page, select a “Token type” (currently there is only one type to choose, which is “Free”) and enter a “Token name” (enter any name you like). Next, press the “CREATE TOKEN” button.



A token will then be created. The “Key” it contains will be used as the API key of your application. Don’t close this page yet, as you will use it shortly later.

After getting an API key, let us start a new project by naming it “CalculateExposure_fromMap”.

Step 2. User interface(UI) design

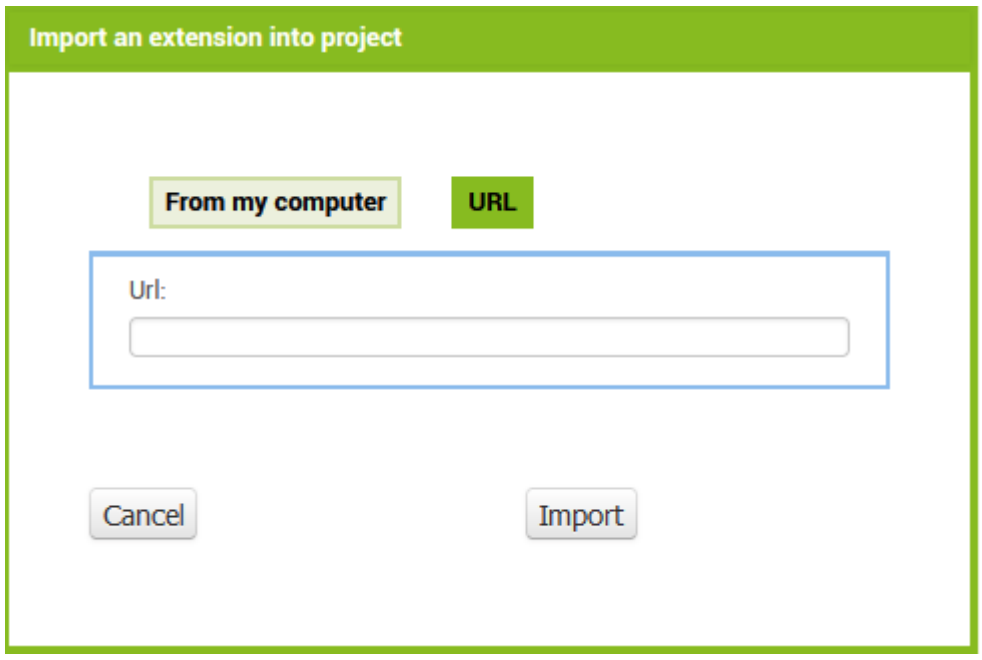
After creating the project, we are automatically in the “Designer” tab.

2.1. Non-visible components:

First we have to import the PRAISE_HK_web extension.

Go to the ‘Palette’ ➔ ’Extension’ and click ‘import extension’

Then, from the box popped up, click the ‘URL’ button, and then input the following URL -- https://envf.ust.hk/~stcheng/PRAISE_HK_web/hk.ust.praise.web_v0.9.aix into the ‘Url’ textbox to import the extension.



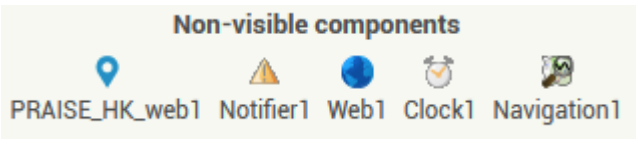
If the import is successful, the ‘Extension’ section will turn into this:

Now, you should drag this onto the phone (like other components). And the extension is ready for use.

Next, we drag other non-visible components in the “Palette” column to the phone too.

1. ‘User Interface’ ➔ ’Notifier’ (*this component displays various pop-up alert messages when the app needs to alert the user for some reason*)
2. ‘Connectivity’ ➔ ’Web’ (*this component enables the app to get/send data from/to the Web*)
3. ‘Sensors’ ➔ ‘Clock’ (*this component provides functionality of a clock*)
4. ‘Maps’ ➔ ‘Navigation’ (*this component is used for route planning*)

The Non-visible components (at the bottom of the phone) should look like the following when completed:



After adding the “Navigation” component, go to its “Properties” column, and fill in its “ApiKey” field, using the key value you have just obtained at the “openroute service” website above.

With this, you can now use the “openroute service” for navigation for free (with quota, please refer to their website for details).

2.2. Visible components:

Firstly, we click on the screen to choose the “Screen1” component, then go to the “Properties” column of “Screen1”, uncheck the “TitleVisible” checkbox. We don’t need to show the title on the screen as this is a single screen application.

2.2.1. Top Area

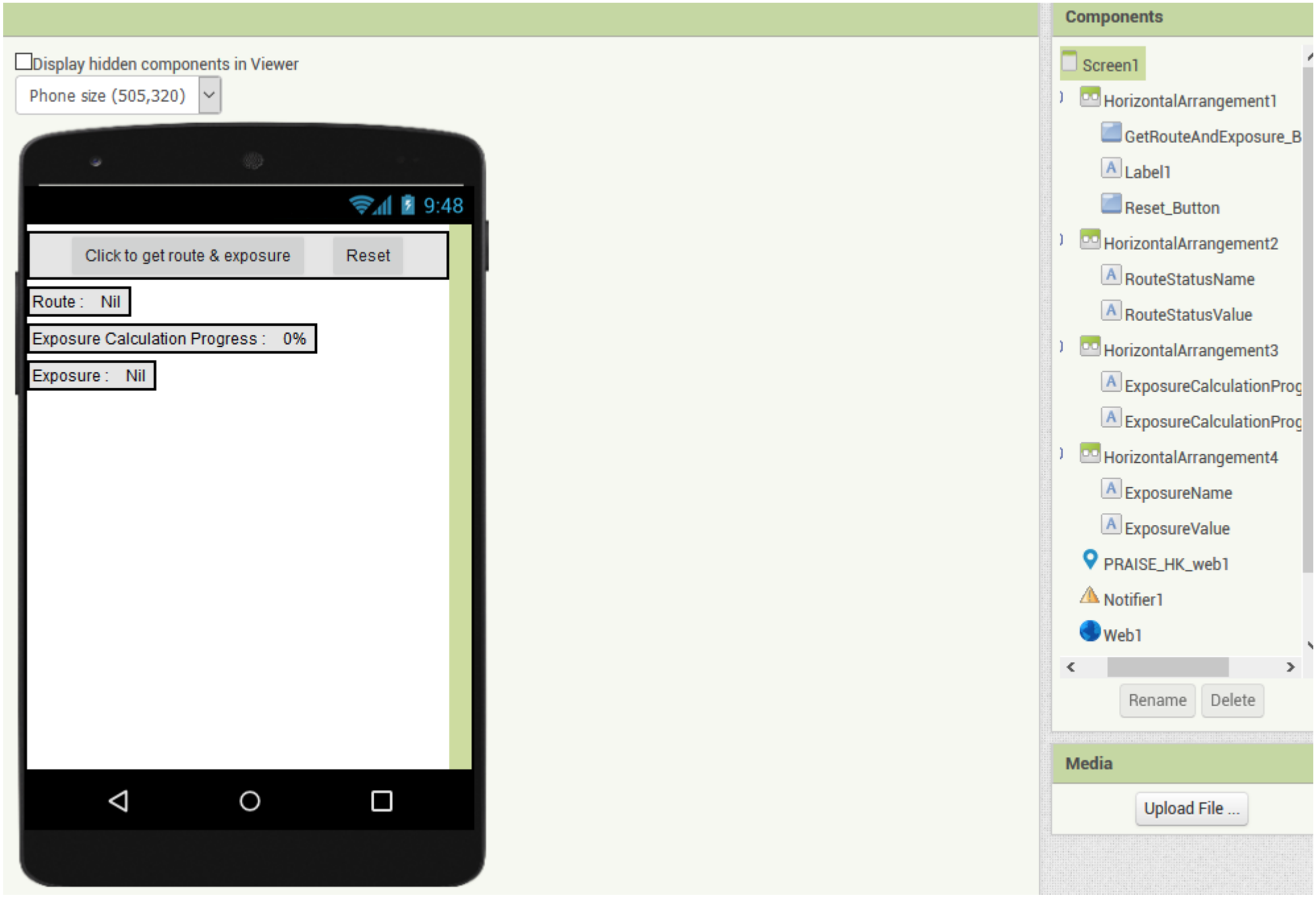
In this area, we drag and drop the various visual components onto “Screen1”, and set the “Properties” as suggested in the following table:

Component Type[1]	***Inside Which Component***	***Follow Which Component***	***Component Name[2]***	***Change Which Properties of the Component***
Layout / HorizontalArrangement	Screen1		HorizontalArrangement1	**AlignHorizontal:** Center **Width:** Fill parent
User Interface / Button	HorizontalArrangement1		GetRouteAndExposure_Button	**Enabled:** Unchecked **Text:** Click to get route & exposure
User Interface / Label	HorizontalArrangement1	GetRouteAndExposure_Button	Label1	**Text:** *{(empty)}*[3]
User Interface / Button	HorizontalArrangement1	Label1	Reset_Button	**Enabled:** Unchecked **Text:** Reset
Layout / HorizontalArrangement	Screen1	HorizontalArrangement1	HorizontalArrangement2	
User Interface / Label	HorizontalArrangement2		RouteStatusName	**Text:** Route :
User Interface / Label	HorizontalArrangement2	RouteStatusName	RouteStatusValue	**Text:** Nil
Layout / HorizontalArrangement	Screen1	HorizontalArrangement2	HorizontalArrangement3	
User Interface / Label	HorizontalArrangement3		ExposureCalculationProgressName	**Text:** Exposure Calculation Progress :
User Interface / Label	HorizontalArrangement3	ExposureCalculationProgressName	ExposureCalculationProgressValue	**Text:** 0%
Layout / HorizontalArrangement	Screen1	HorizontalArrangement3	HorizontalArrangement4	
User Interface / Label	HorizontalArrangement4		ExposureName	**Text:** Exposure :
User Interface / Label	HorizontalArrangement4	ExposureName	ExposureValue	**Text:** Nil

Annotation —

1. Component Type are items in the “Palette” column
2. if needed, click the “Rename” button in the “Components” column to change the name of a particular component
3. {(empty)} means the property is really empty, devoid of any content.

If everything is alright, the appearance would look like this:



2.2.2. Bottom Area (Map Area)

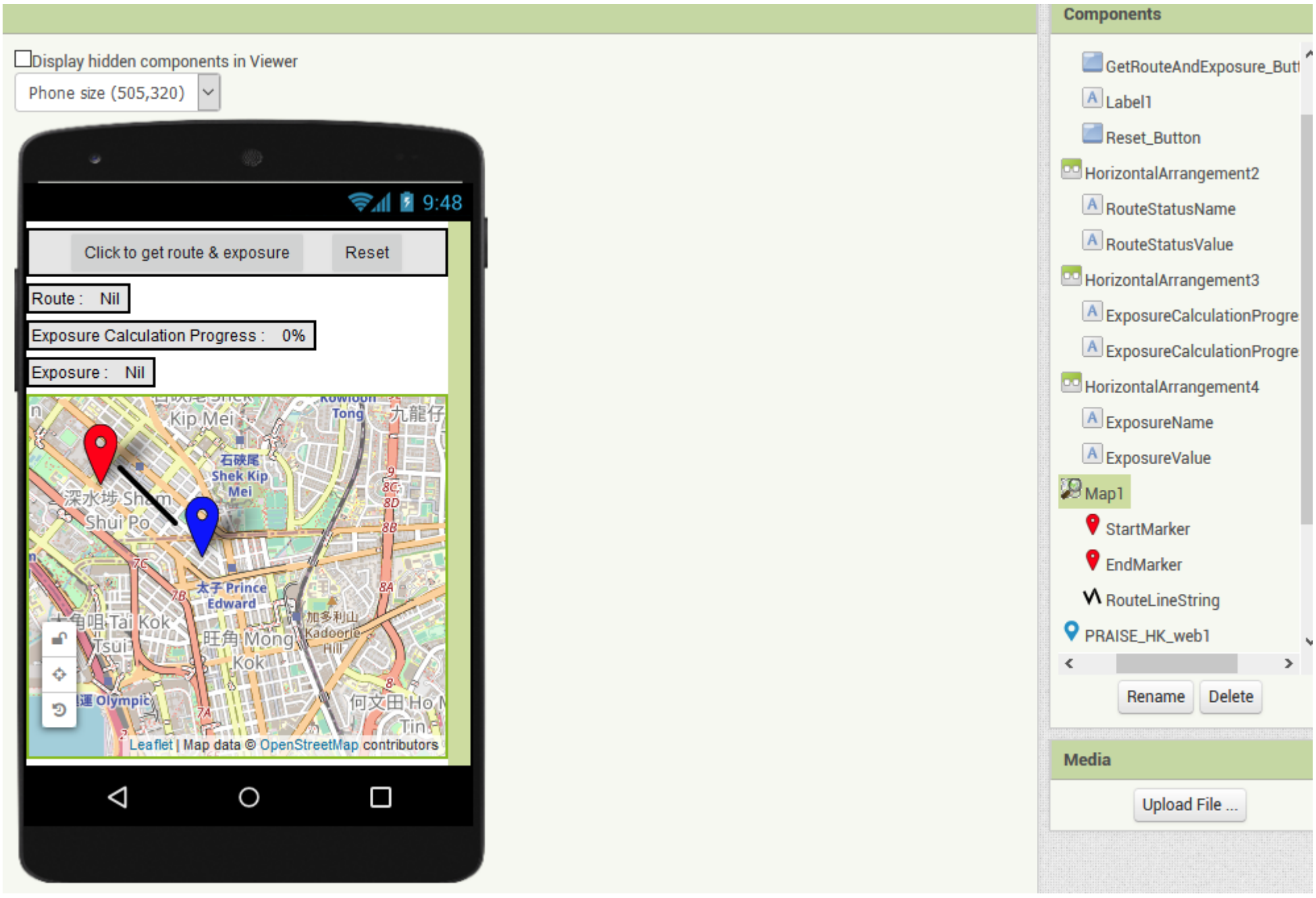
In this area, we first need to drag and drop a map component onto “Screen1”, under the top area, and set the “Properties” as suggested in the following table:

Component Type	***Inside Which Component***	***Follow Which Component***	***Component Name***	***Change Which Properties of the Component***
Maps / Map	Screen1	HorizontalArrangement4	Map1	**CenterFromString:** 22.324058, 114.168601 **Height:** Fill parent **Width:** Fill parent **ZoomLevel:** 14
Maps / Marker	Map	{{*Follow no one, just drag the component onto the map is alright*}}	StartMarker	**Latitude:** 22.3295141[1] **Longitude:** 114.1596221[1] **Visible:** Unchecked
Maps / Marker	Map	{{*Follow no one, just drag the component onto the map is alright*}}	EndMarker	**Latitude:** 22.3251473[1] **Longitude:** 114.1662311[1] **Visible:** Unchecked
Maps / LineString	Map	{{*Follow no one, just drag the component onto the map is alright*}}	RouteLineString	**StrokeWidth:** 4 **Visible:** Unchecked

Annotation —

1. Free to choose any points on the map, those values are only suggestion

If everything is alright, the appearance would look like this:



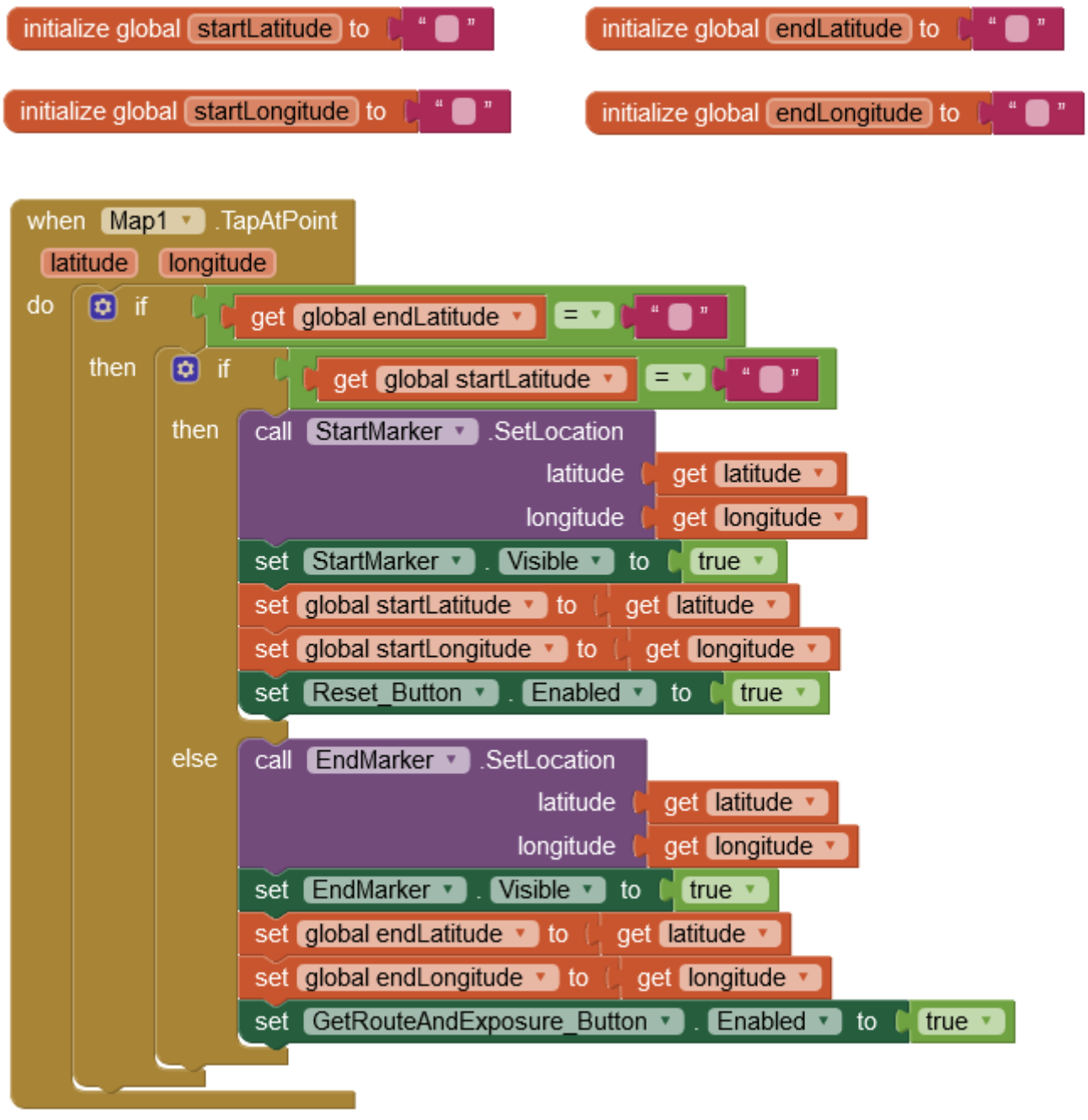
The UI is now completed. Next step will be the implementation of the behaviour.

Step 3. Behaviour Implementation (Blocks Building)

First switch to the Blocks Editor.

3.1. Map1 (Bottom Area)

We use the map to get the start and end points of the route, so we assign the following block structure to the “TapAtPoint” event handler of the “Map1” (as there are global variables inside this structure, we thus need to initialize them first) :



Explanation:

The global variables are used as follows:

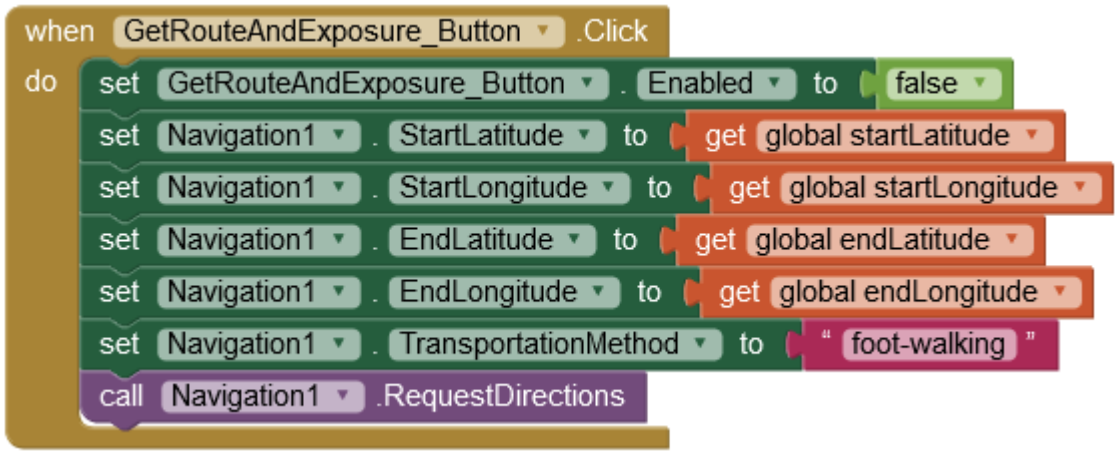
- “startLatitude” and “startLongitude” is used to store the current start point chosen by the user
- “endLatitude” and “endLongitude” is used to store the current end point chosen by the user

At the beginning, all are set to an empty string, meaning that nothing is stored in them.

If the user taps any point on the map, the above “Map1.TapAtPoint” event handler will be run, and passed in the latitude and the longitude value of that point. The two “if” statements check if start and/or end points are already chosen by the user. According to the status (which points are chosen), respective actions are performed (such as placing markers to the right places, enabling some buttons).

3.2. GetRouteAndExposure_Button (Top Area)

We click this button to get the route and the corresponding exposure. When clicked, the “GetRouteAndExposure_Button.Click” event handler will be called, and we will use it to get the route first:

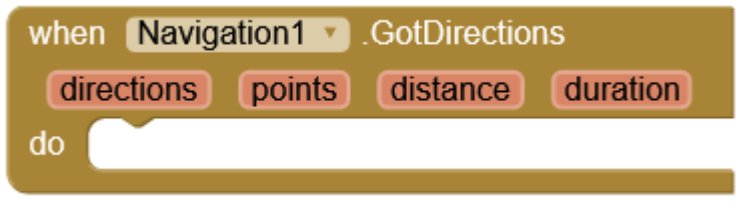


Explanation:

Mainly, we use the “Navigation” component to send the direction request in this block structure. First, we set the start and end point (chosen by the user in the previous step) using the respective “Navigation” component’s properties. Also, set its “TransportationMethod” to “foot-walking” (so that makes it compatible with the result we got from the exposure calculation of PRAISE-HK API later). Finally, call the “Navigation1.RequestDirections” to send the request to the service.

3.3. Navigation1 (Non-visible Component)

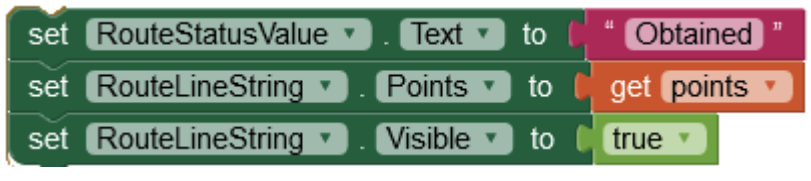
When the service returns the navigation directions, the “Navigation1.GotDirections” event handler is called.



The block structure is quite large, so it is disassembled and introduced part by part.

3.3.1.

First, add the following blocks to the event handler:



Explanation:

The first block: “RouteStatusValue.Text” is set to “Obtained” to indicate a route is successfully obtained. The other two blocks simply draw the route out.

3.3.2.

Second, add the following block structure (with a global variable which is initialized outside it):
[Note: here you just need to add the global variable. For other blocks, you will be told when to add them in the explanation below.]

Explanation:

In order to calculate the exposure, for example, we need to know the locations along the route (these successive locations cut the route into “segments”), and how long to travel between locations (so that the duration of each segment). That information is located in “Navigation1.ResponseContent”. This property populates the raw data returned by the service. It is in JSON format (a very popular format for data exchange between devices on the web). App Inventor converts it into a dictionary (the conversion works because conceptually JSON and App Inventor’s dictionary are pretty much the same thing).

Now, let’s look an example of what information is contained in this dictionary:

```
{
   ........,

   "features":[
      {
         ........,

         "properties":{
            "segments":[
               {
                  "distance":2570.8,
                  "duration":259.5,
                  "steps":[
                     {
                        "distance":175.8,
                        "duration":11.5,
                        "type":11,
                        "instruction":"Head northwest on 元州街 Un Chau Street",
                        "name":"元州街 Un Chau Street",
                        "way_points":[
                           0,
                           3
                        ]
                     },
                     {
                        "distance":64.9,
                        "duration":15.6,
                        "type":1,
                        "instruction":"Turn right onto 發祥街 Fat Tseung Street",
                        "name":"發祥街 Fat Tseung Street",
                        "way_points":[
                           3,
                           4
                        ]
                     },

                     ........,

                     {
                        "distance":0.2,
                        "duration":0.2,
                        "type":0,
                        "instruction":"Turn left onto 太子道西 Prince Edward Road West",
                        "name":"太子道西 Prince Edward Road West",
                        "way_points":[
                           61,
                           62
                        ]
                     },
                     {
                        "distance":0,
                        "duration":0,
                        "type":10,
                        "instruction":"Arrive at 太子道西 Prince Edward Road West, straight ahead",
                        "name":"-",
                        "way_points":[
                           62,
                           62
                        ]
                     }
                  ]
               }
            ],
            ........,
         },
         "geometry":{
            "coordinates":[
               [
                  22.336381,
                  114.158015
               ],
               [
                  22.336706,
                  114.157582
               ],
               [
                  22.337037,
                  114.157163
               ],
               [
                  22.3374,
                  114.156709
               ],
               [
                  22.337855,
                  114.157104
               ],

               ........,

               [
                  22.323735,
                  114.166733
               ],
               [
                  22.323736,
                  114.166735
               ]
            ],
            "type":"LineString"
         }
      }
   ],
   ........
}
```

Please be noted that some of the information is deliberately left out as it is irrelevant to the issue we are dealing with currently.

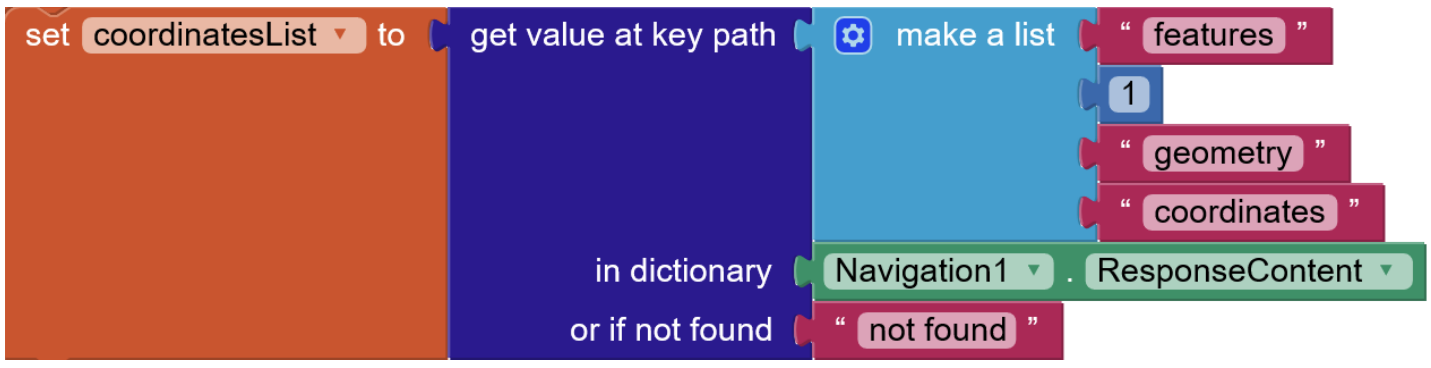
Firstly, you may notice there is a “segments” list to which the path is: ***Dictionary***[“features”] ➔ ***List***[1] ➔ ***Dictionary***[“properties”] ➔ ***Dictionary***[“segments”]. But indeed it is the “steps” list inside the “segments” list that contains the actual segments. The path to it is: ***Dictionary***[“features”] ➔ ***List***[1] ➔ ***Dictionary***[“properties”] ➔ ***Dictionary***[“segments”] ➔ ***List***[1] ➔ ***Dictionary***[“steps”]. Now let’s take a look at each step. Inside each step, there is a property called “duration”, and also another property called “way_points”. The “way_points” contains a pair of points inside, astute readers may ponder they are corresponding to start and end points respectively. Yes, they are right. But what does point 0 and 3 mean? The answer lies at the “coordinates” property to which the path is: ***Dictionary***[“features”] ➔ ***List***[1] ➔ ***Dictionary***[“geometry”] ➔ ***Dictionary***[“coordinates”]. There is a list of coordinates(a pair of latitude and longitude) inside it. Point 0 is the first one in the list, and point 3 is the fourth one.

Armed with this new knowledge, we are finally in position to understand the block structure above. *And from now please start to add the blocks to the editor as they appear.*

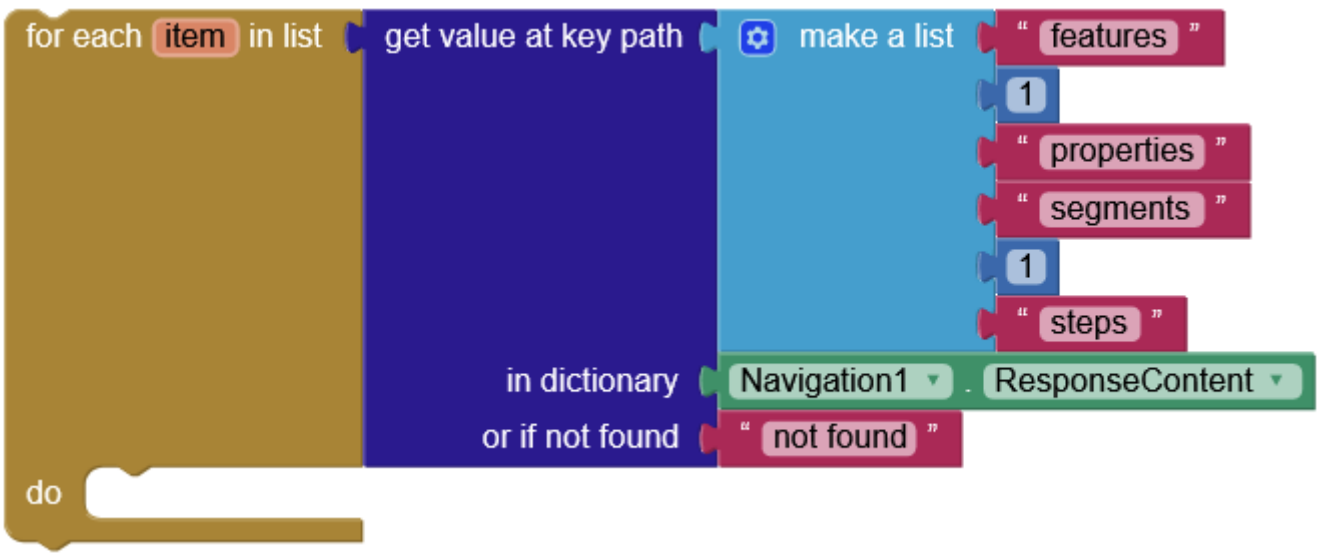
It first initializes the local variables that will be used.

Note: all the blocks below should be placed inside the “in” section of this block

Next, we populate the “coordinatesList” variable with the items inside the “coordinates” property (this is why it is called “coordinatesList”). In order to do this, we use one of the dictionary functions: “get value at key path” (click the link to pay a visit to the reference if you don’t know what it is). As described in the last paragraph, the path should be: “features”, 1, “geometry”, “coordinates”, and it is indeed displayed exactly as it is in the block structure above.

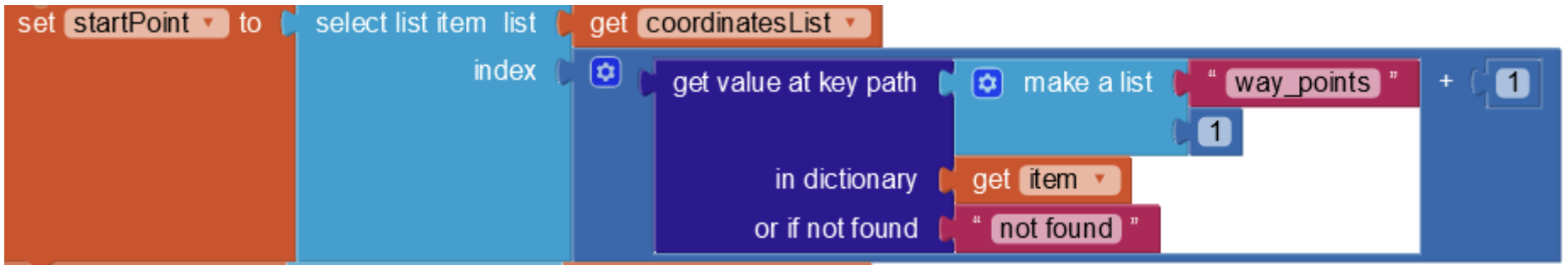


After setting up the “coordinatesList”, we can now deal with extracting data from the “segments”. Like the “coordinatesList”, the “segments” (steps) list can be obtained using the same dictionary function, with the path: “features”, 1, “properties”, “segments”, 1, “steps”. We want to loop through this list to get details of each segment, so we pass it to a “for” control block.



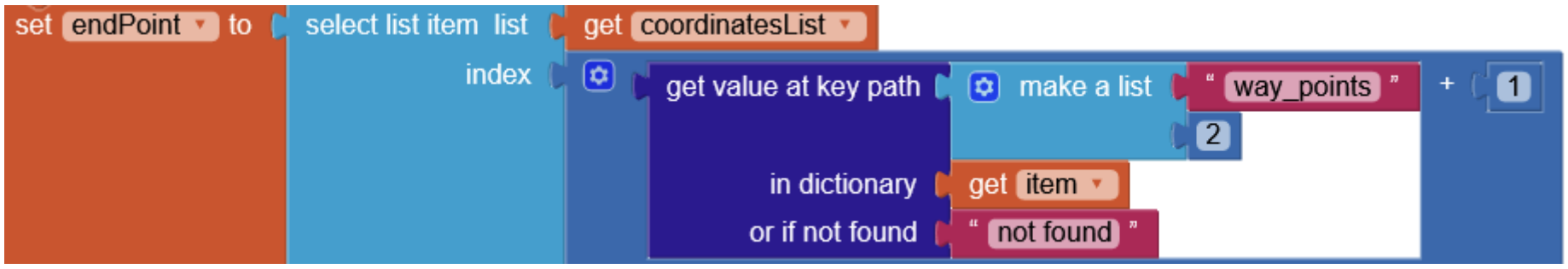
Note: all the blocks below should be placed inside the “do” section of this block

Each “item” in the “for” loop represents a “segment”. And for each “segment”, we get and set the “startPoint” with the following blocks:



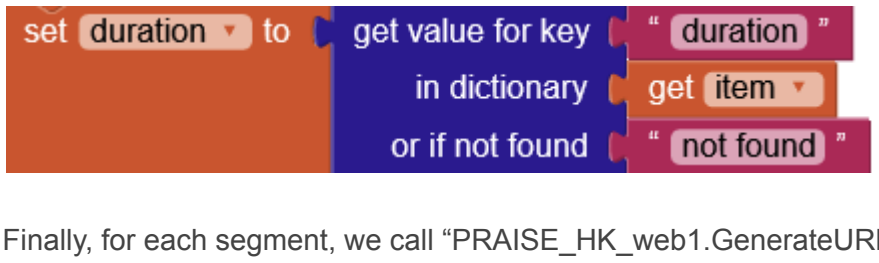
The “startPoint” coordinates are actually stored in the “coordinatesList”, but we can use the “way_points” property in the “segment” (“item” variable) to get them. Beware that point 0 in the “way_points” means index 1 in the “coordinatesList”, so we have to “+1” to get the correct index.

For the “endPoint”, the procedure to get it is similar to the “startPoint”.

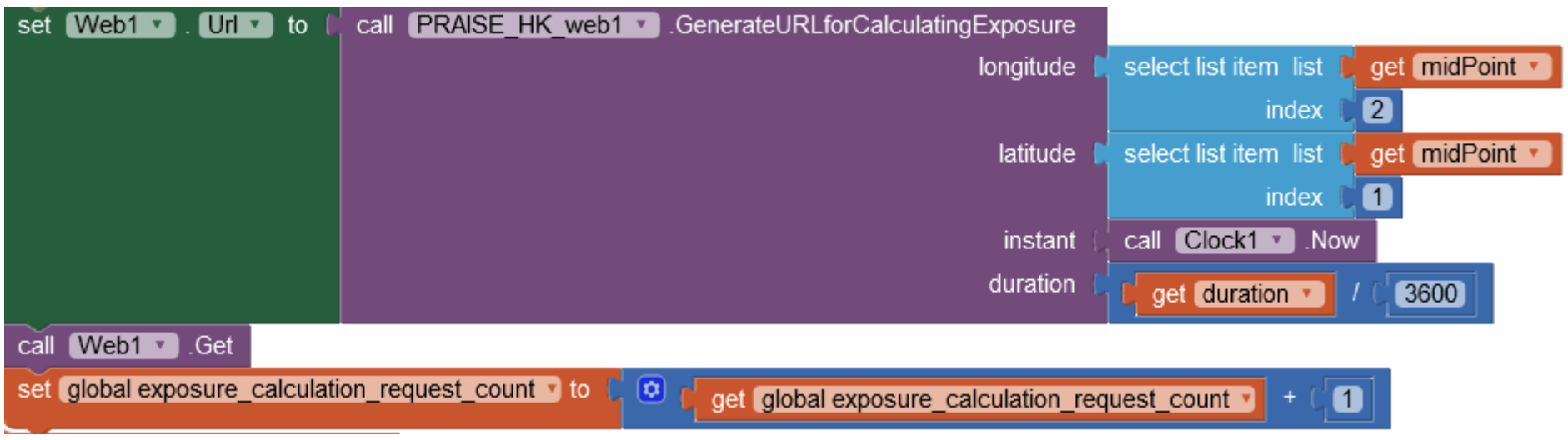


To calculate the exposure, we have to pick a point in the segment for approximation. And usually the most reasonable approximation is the midpoint. To get the midpoint, we just need to calculate the average of the latitude and longitude, as follows:

Now, we have a position (midpoint) to represent the segment, we also need its duration:

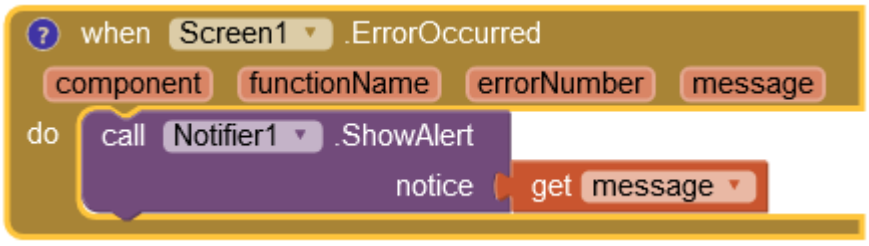


Finally, for each segment, we call “PRAISE_HK_web1.GenerateURLforCalculationgExposure” to generate the correct URL. We put the “midPoint” & “duration” we got above into respective parameters. Be also noted that the current time instant (by calling “Clock1.Now ") is put into the “instant” parameter, so the exposure we get is a real-time exposure. And the “duration” variable we have is in unit of “second” while the “duration” parameter only accepts unit of “hour”, so we convert it by dividing 3600. After the URL is set, we send the request using the “Web1.Get” method. And add one to the global “exposure_calculation_request_count” variable, for recording down how many web requests are made in total.



3.3.3.

There is a rare situation when the “Navigation” component fails to get directions. When this happens, the “Screen1.ErrorOccured” is called.



Explanation:

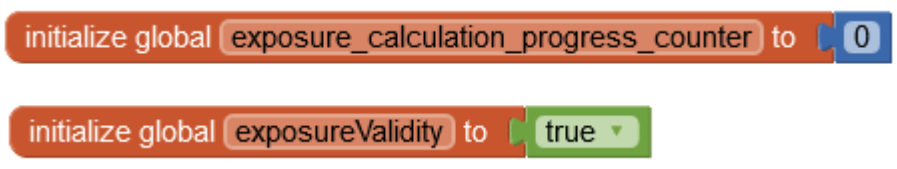
Inside the handler, we simply call the “Notifier1.ShowAlert” to show the error message (got from the input argument) to the user.

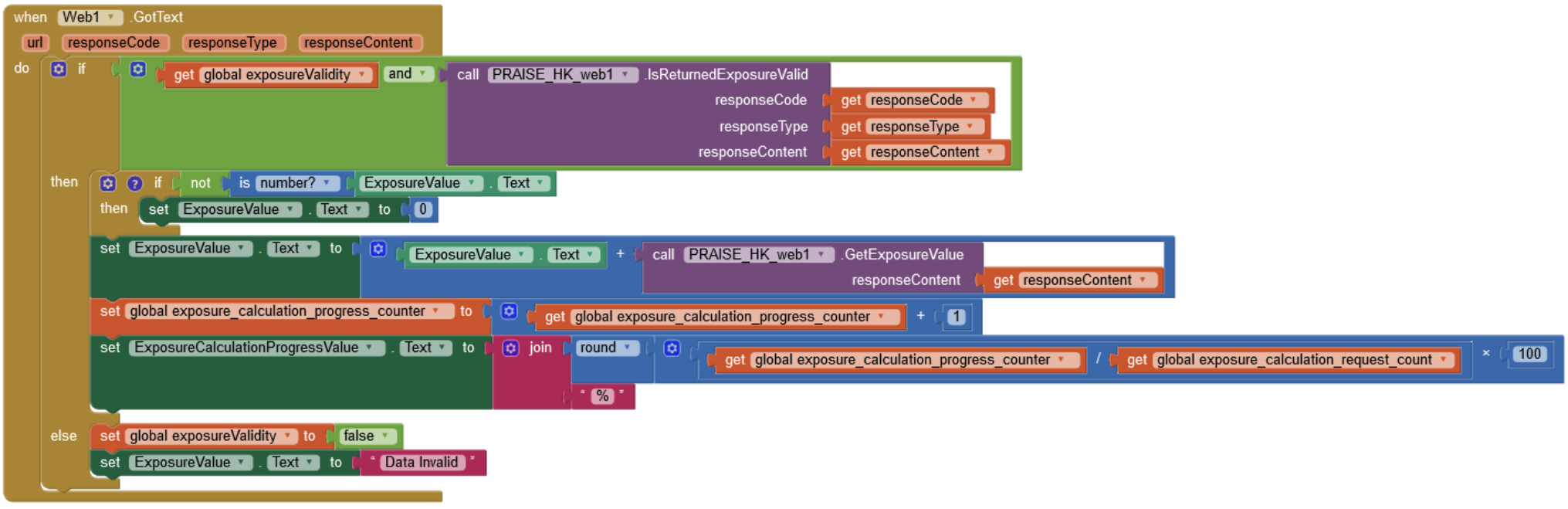
3.4. Web1 (Non-visible Component)

3.4.1.

When there is a response from the web, the “Web1.GotText” event handler will be called. So, we use the following structure to deal with it (with a pair of initialized global variables it depends on):

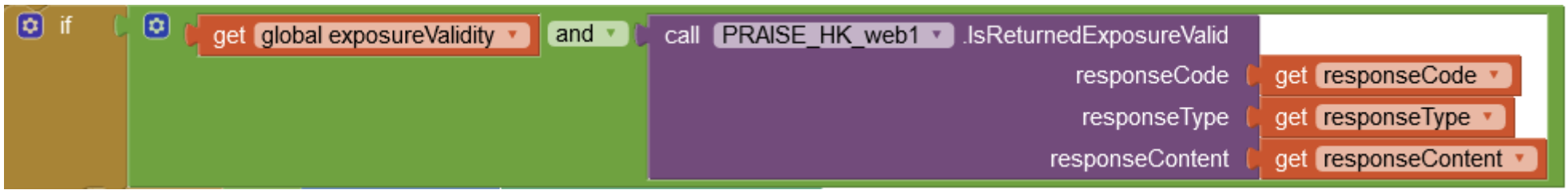
**** Note: This handler will be called multiple times as we have made multiple web request***





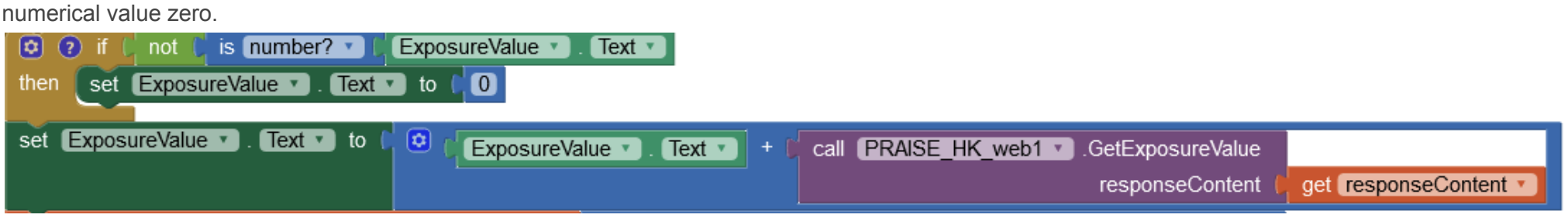
Explanation:

Inside this handler, we first check (the first “if” statement) if both the value in the “exposureValidity” is true and the result returned by calling the “PRAISE_HK_web1.IsReturnedExposureValid” method is also true. This method checks if the returned response is indeed a valid one. We just need to put all elements of the response into it (i.e. the response code, type and content), then you will be given the answer. If this is the first time the handler being called, the “exposureValidity” must be true (because it is initialized to be true), then only the calling result needs to be considered.

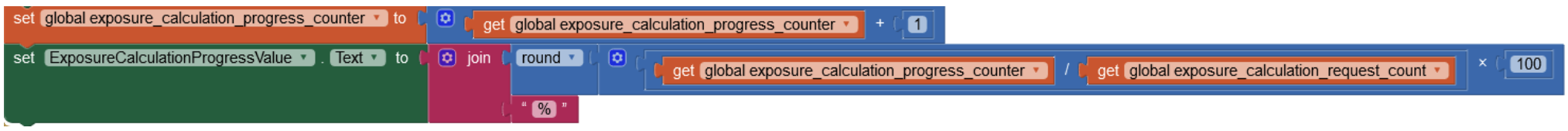


But if somehow the call fails even just once, the “else” section will be executed. Inside the “else” section, the “exposureValidity” is set to false. With it set to false, all subsequent “if” statement checks will be false automatically, and the calculation process stops. So, what does this mean? This means just one incorrect response can make the whole exposure calculation invalid.

Anyway, if the first “if” statement is passed, we can begin or continue the exposure calculation process. The calculation is simple, we just add the accumulated value (which is stored in “ExposureValue.Text”) to the new value got in the current response (the result returned by calling the “PRAISE_HK_web1.GetExposureValue” method). Please be noted that if this is the first time the handler is called, the value inside “ExposureValue.Text” is “Nil”, which is not a number and cannot be used for addition. This is why the second “if” statement exists. It converts the string value “Nil” to the numerical value zero.

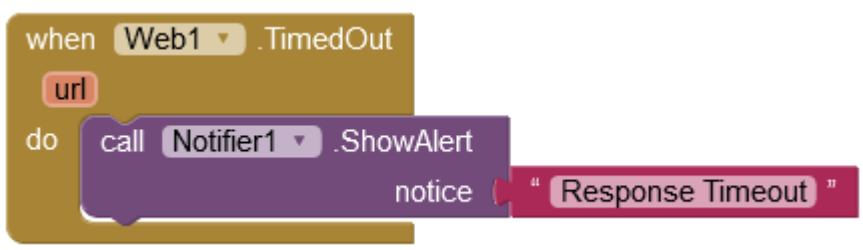


Finally, we update the progress. Here we use two global variables: “exposure_calculation_progress_counter” & “exposure_calculation_request_count” to record the progress. And then display it using the “ExposureCalculationProgressValue.Text”. This value is shown in unit of “percentage”, so we have to perform some simple mathematical conversion.



3.4.1.

There is a rare situation when there is no response from the web. We handle this using the “Web1.TimeOut” event handler.

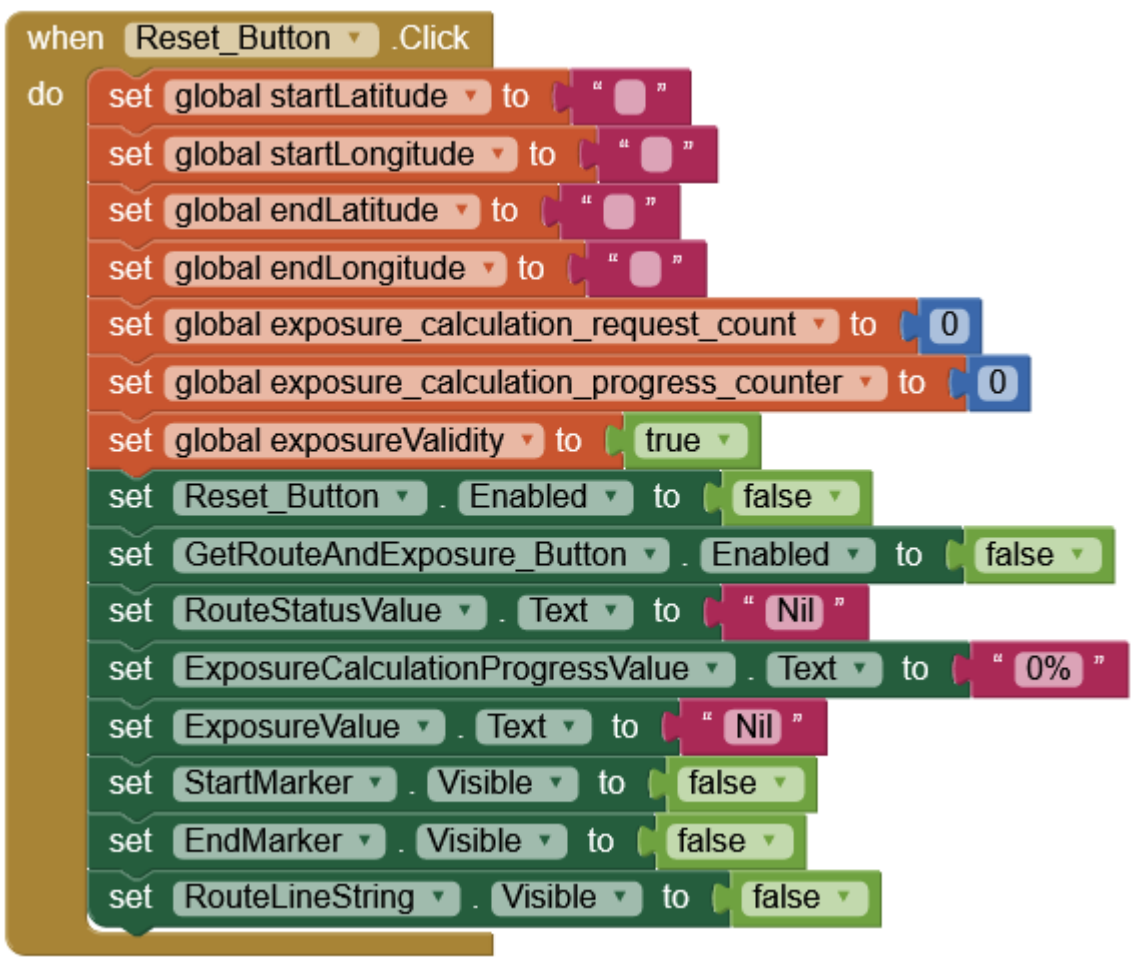


Explanation:

Inside the handler, we simply call the “Notifier1.ShowAlert” to tell the user there is no response from one of the requests.

3.5. Reset_Button (Top Area)

We click this button to reset the app to its initial state. When clicked, the “Reset_Button.Click” event handler will be called, and the blocks for resetting are inserted:



Explanation:

Nothing complex here, all the components (both visible and non-visible) and the global variables are reset to their initial states, if they are set to another state during the process.

Conclusion

This tutorial serves as a proof that PRAISE-HK service can be used with App Inventor. So, now even average secondary students or non-coders can produce air quality aware apps!